Fundamentos de Python

Lección 1: Historia de los lenguajes de programación. Filosofía de Python.

eip ESCUELA INTERNACIONAL POSGRADOS

ÍNDICE

Historia de los lenguajes de programación.

Presentación y objetivos

En este capítulo haremos un repaso por la historia de los diferentes lenguajes de programación, analizando cómo han evolucionado para adaptarse a la necesidad de cada época. En concreto, estudiaremos desde los primeros lenguajes, donde se utilizaban tarjetas perforadas para indicar una serie de instrucciones a grandes máquinas calculadoras, hasta los lenguajes de programación más recientes, como Python, Java o C++.

Objetivos

- **Conocer cuáles fueron los orígenes de los lenguajes de programación.**
- **Conocer las diferentes generaciones de lenguajes de programación.**
- **Comprender la tipología de los lenguajes de programación.**
- **Comprender las diferencias entre los lenguajes compilados e interpretados.**
- **Conocer la historia y filosofía de Python.**

1 INTRODUCCIÓN A LOS LENGUAJES DE LA PROGRAMACIÓN

A lo largo de la historia se ha perseguido el desarrollo de máquinas que ayudaran a la humanidad a realizar tareas mecánicas y tediosas de una forma más sencilla. Esta historia va desde el siglo XVII con el desarrollo las primeras calculadoras mecánicas, tales como la **máquina de Schickard** en 1623 o la **calculadora de Pascal** en 1650, hasta las máquinas y computadoras modernas que conocemos hoy en día.

Es importante destacar que, a lo largo de la evolución de las computadoras, existe una en concreto que marcó un cambio en la forma de entender y manejar estas máquinas. Hablamos de la famosa **máquina analítica** diseñada por **Charles Babage** en 1840. La principal diferencia entre esta máquina y el resto de artefactos es que la máquina analítica fue la **primera máquina programable** mediante tarjetas perforadas (ver Imagen 1). En concreto, la máquina permitía que los cálculos realizaran saltos condicionales y bucles.

Presta atención

El primer lenguaje de programación surgió en 1843 de mano de Ada Lovelace. Colaboró con Charles Babbage en el desarrollo de lo que podría considerarse el primer ordenador mecánico: la máquina analítica. Lovelace fue la primera persona en implementar un algoritmo para la máquina analítica por lo que, gracias a esta contribución, se ganó el título de primera programadora del mundo.

Un siglo después, en 1936, el matemático inglés **Alan Turing** definió de forma abstracta el computador, utilizando un modelo muy sencillo de procesamiento: una máquina capaz de leer y escribir ceros y unos en una cinta infinita (que actuaba como memoria), moviéndose y escribiendo a lo largo de la cinta siguiendo una secuencia de instrucciones (un programa). A dicha máquina se la conoce como **máquina de Turing.**

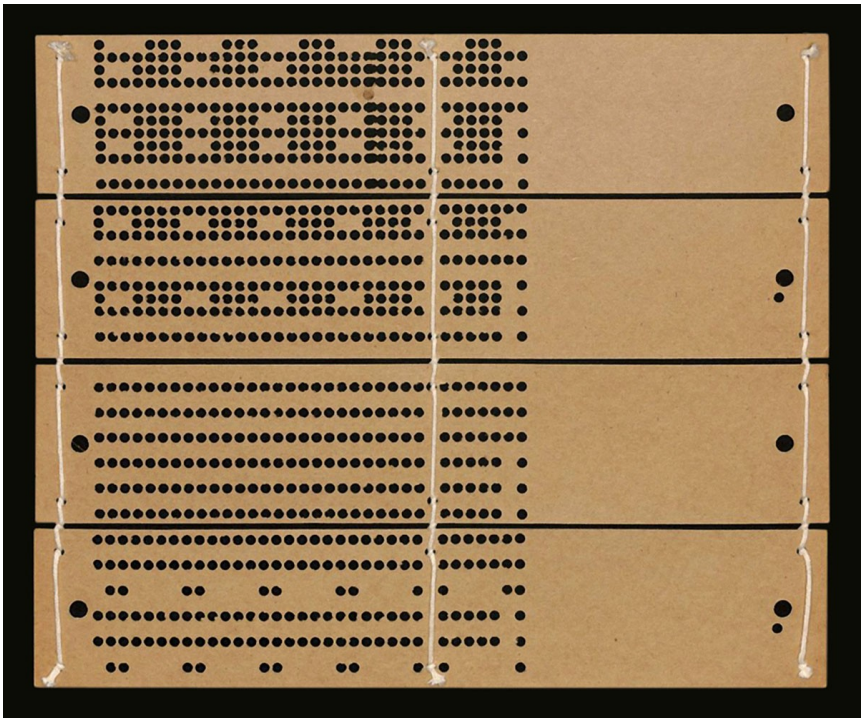


Imagen 1: Tarjeta perforada del telar mecánico de Jacquard.

Posteriormente, en 1945 se introdujo un avance fundamental que sentaría las bases del primer computador electrónico digital. Hablamos de la arquitectura Von Neumann (en honor a su creador, **John Von Neumann**) en la que por primera vez se definen dos ideas claves de los computadores de propósito general: que un **programa** pueda ser **almacenado en memoria** y un conjunto de **instrucciones de procesamiento que permita cargar y ejecutar un programa**. Gracias a este hito, en 1948, se construyó en la Universidad de Manchester el primer computador electrónico digital.

Conceptos

Un lenguaje de programación se puede definir como un vocabulario y un conjunto de reglas gramaticales para dar instrucciones a una máquina para que ésta realice un conjunto de tareas específicas. En términos generales, un lenguaje de programación es una herramienta que nos permite interactuar con una máquina, a fin de indicarle una secuencia de tareas que deben ser realizadas.

A partir de estos avances en el desarrollo de computadores cada vez más potentes, comenzaron a crearse programas diseñados a resolver tareas específicas (lo que hoy en día conocemos como software). Estos programas son escritos o desarrollados en un determinado lenguaje de programación. Como hemos estudiado, los lenguajes de programación surgieron y han evolucionado desde el nacimiento de los primeros computadores, en los que al inicio se utilizaba código máquina para indicarle una serie de instrucciones que un computador había de resolver. Esté código máquina tiene las siguientes características:

- Es muy lejano del lenguaje natural.
- La sintaxis es muy estricta.
- Las operaciones que realiza son muy limitadas y elementales.
- Es distinto para cada tipo de CPU.
- Está muy ligado al hardware.

Para solucionar este problema, aparecen **los lenguajes de programación de alto nivel**. Estos fueron desarrollados para facilitar la comprensión e interacción entre los usuarios y la máquina. Las principales ventajas de los lenguajes de alto nivel son:

- Fáciles de interpretar.
- Sintaxis definida de una forma clara.
- Adecuado para empresas o aplicaciones científicas.
- Es independiente de la máquina que lo utilice.
- Portabilidad a otras plataformas.
- Fácil de depurar.
- Alta velocidad de ejecución.

De este modo, surgieron diferentes lenguajes de programación a lo largo de la historia, todos con el propósito de crear una herramienta que ayudara al usuario a transmitir una secuencia de instrucciones que el ordenador debe ejecutar. En concreto, en la historia de los lenguajes de programación podemos identificar 5 generaciones de estos lenguajes:

La **primera generación** de los lenguajes de programación consistía en lenguajes a nivel de máquina; se comunicaban por medio de ceros y unos. Como podéis imaginar, el principal inconveniente de este tipo de lenguajes de programación es que escribir un programa en código máquina es una tarea ardua, además de estar sujeto a errores.

Esto implicó que el aprendizaje de este lenguaje fuese una tarea muy difícil, además de que el código debía ser diseñado de forma específica para que pudiera ser ejecutado en una determinada máquina. Si era necesario utilizar ese código desarrollado en otra máquina, este debía ser reescrito y adaptado para ser ejecutado.

Importante

La principal ventaja de los lenguajes de primera generación es su alta velocidad de ejecución, ya que se ejecutan directamente en la CPU del ordenador. Es decir, estos lenguajes NO necesitan de un compilador para convertir el lenguaje de programación a código máquina.

Los lenguajes de programación de **segunda generación** hacen referencia al lenguaje ensamblador, el cual era específico para un ordenador y procesador específico. Aun así, estos lenguajes eran más interpretables que los lenguajes de primera generación. Un programa escrito en ensamblador solo podrá ser ejecutado en una familia específica de procesadores, lo que implica que un programa desarrollado con este lenguaje no puede ser ejecutado en otra familia de procesadores. Los lenguajes de segunda generación siguen siendo utilizados hoy en día, aunque los lenguajes de alto nivel los han reemplazado.

Los lenguajes de **tercera generación** incluyen lenguajes como FORTRAN, C o PASCAL. Son lenguajes de propósito general y han sido utilizados en ámbitos empresariales y científicos. Algunos ejemplos de lenguajes de tercera generación son FORTRAN, COBOL, C++ o Java. Las principales ventajas de estos lenguajes de programación son:

- Alta interpretabilidad
- Sintaxis claramente definida.
- Útil para el desarrollo de aplicaciones científicas y empresariales.
- Independiente del computador y procesador.
- Fácil de depurar y corregir errores.
- Alta velocidad de ejecución.

Importante

Los lenguajes de tercera generación pueden ser ejecutados en cualquier computador, independientemente de su arquitectura y procesador.

Puesto que estos lenguajes han marcado un antes y un después en la forma de entender los lenguajes de programación, cabe destacar que los primeros lenguajes de programación de alto nivel surgieron a finales de la década de los 50, y fueron **FORTRAN** en 1956 y **Lisp** en 1958.

- FORTRAN fue desarrollado por **John Backus** y se le considera como el lenguaje de programación más antiguo que sigue utilizándose hoy en día. Entre sus características podemos destacar que es un **lenguaje imperativo** y **compilado**. En la siguiente imagen podemos ver un ejemplo de un programa realizado en FORTRAN para resolver una operación matemática.

Example: It is required to compute the following quantities

$$P_i = \sqrt{\sin^2 (A_i B_i + C_i) + \cos^2 (A_i B_i - C_i)}$$

$$Q_i = \sin^2 (A_i + C_i) + \cos^2 (A_i - C_i)$$

for i = 1, ..., 100. A possible FORTRAN program for this calculation follows.

C ← FOR COMMENT / STATEMENT NUMBER	CONTINUATION	FORTRAN STATEMENT
1		TRIGF(X,Y) = SINF (X+Y)**2+COSF(X-Y)**2
2		DIMENSION A(100), B(100), C(100), P(100), Q(100)
3		READ 8, A, B, C
4		DO 6 I = 1,100
5		P(I) = SQRTF(TRIGF(A(I)*B(I), C(I))
6		Q(I) = TRIGF(A(I), C(I))
7		PRINT 8, (A(I), B(I), C(I), P(I), Q(I), I = 1,100)
8		FORMAT (5F 10.4)
9		STOP

Imagen 2: Ejemplo de código FORTRAN (Manual de FORTRAN del IBM 704)

El lenguaje de programación Lisp fue desarrollado por **John McCarty** en el Instituto de Tecnología de Massachusetts (MIT). En sus orígenes, Lisp fue propuesto para desarrollar labores de Inteligencia Artificial y es uno de los lenguajes de programación más antiguos que siguen utilizándose y que pueden ser utilizados en lugar de Ruby o Python. Prueba de ello, importantes compañías como Acceleartion, Boeing y Genworks siguen haciendo uso de Lisp. A diferencia de FORTRAN, Lisp es un **lenguaje funcional** e **interpretado**. En la siguiente imagen podemos ver un ejemplo de un programa implementado con el lenguaje de programación LISP para calcular el tamaño de una lista de forma recursiva.

3. The Function LENGTH, defined Recursively

If we want to define LENGTH recursively, the expression is even simpler and easier:

To find the LENGTH of a list M:

If the list is null, take 0. Else,
Add 1 to the length of CDR M.

The LISP defining expression is:

```
(LENGTH (LAMBDA (M) (COND
0       1       2 2 2
     ((NULL M) 0)
     34      4  3
     (T (PLUS 1 (LENGTH (CDR M))))))
     3  4       5       6     6543210
```

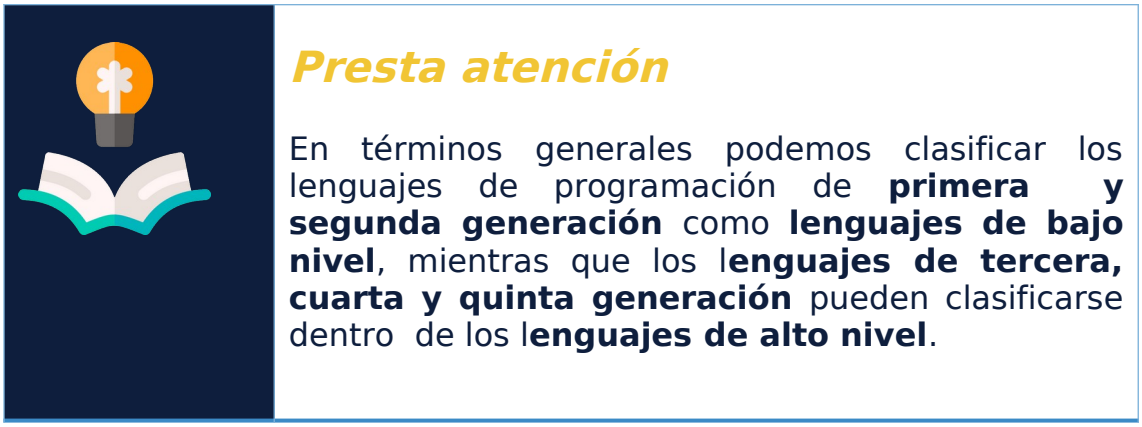
Imagen 3: Ejemplo de código Lisp (The Programming Language LISP, MIT Press)

Los lenguajes de programación de **cuarta generación** fueron diseñados para que el programador especificara qué tenían que hacer, en lugar de como debían hacerlo. Fueron diseñados para reducir el esfuerzo de programar y para incluir generadores de informes y generadores de forma. Los generadores de informes toman una descripción del formato de los datos y el informe que debe ser generado, y automáticamente se genera un programa que produce un informe. Los generadores de informes suelen ser utilizados para generar programas que ayuden en la gestión de interacciones online con los usuarios de las aplicaciones. Sin embargo, los lenguajes de cuarta generación son mucho más ineficientes (en términos de tiempo computacional) que los lenguajes de tercera generación. Los lenguajes de esta generación son utilizados comúnmente en programación de bases de datos y desarrollo de scripts. A continuación, se muestran algunos ejemplos de lenguajes de cuarta generación:

- **Perl**
- **PHP**
- **Pyton**
- **Ruby**
- **SQL**

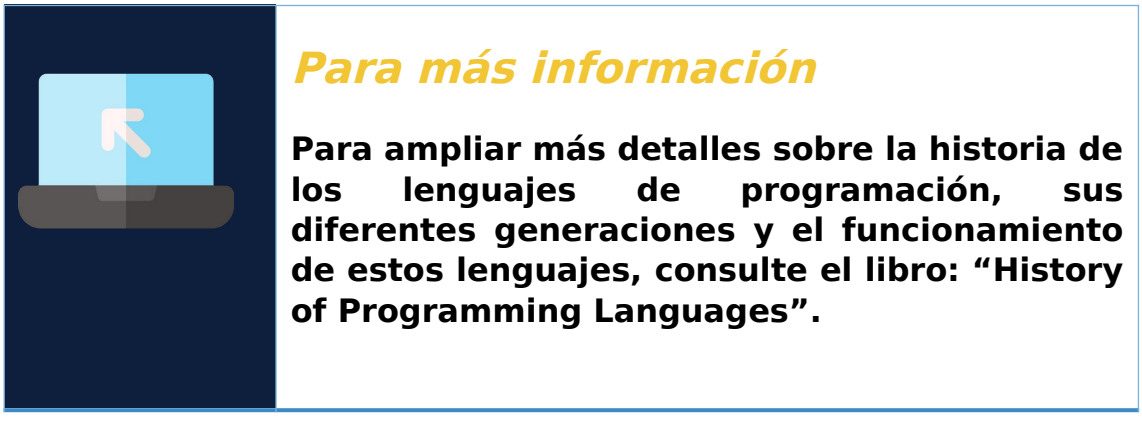
Los lenguajes de **quinta generación** fueron diseñados para resolver problemas utilizando restricciones aplicadas a un determinado programa, en lugar de utilizar un algoritmo escrito por el programador. En otras palabras, estos lenguajes tratan de hacer que el propio computador solucione un problema, en lugar de que se lo indique el programador. En concreto, el programador define el problema y las condiciones que deben ser cumplidas, pero no le indica ningún detalle acerca del algoritmo a utilizar ni del lenguaje de programación. Estos lenguajes son utilizados principalmente en tareas relacionadas en el área de la Inteligencia Artificial. Este tipo de lenguajes son utilizados principalmente en ámbitos académicos. Entre los lenguajes de programación de quinta generación más conocidos podemos encontrar:

- **Prolog**
- **Mercury**
- **OPS5**



Presta atención

En términos generales podemos clasificar los lenguajes de programación de **primera y segunda generación** como **lenguajes de bajo nivel**, mientras que los l**enguajes de tercera, cuarta y quinta generación** pueden clasificarse dentro de los l**enguajes de alto nivel**.



Para más información

Para ampliar más detalles sobre la historia de los lenguajes de programación, sus diferentes generaciones y el funcionamiento de estos lenguajes, consulte el libro: “History of Programming Languages”.

Tras este breve repaso por la historia de los lenguajes de programación y las diferentes generaciones de estos que han surgido, cabe destacar que desde el año 1954 hasta la actualidad se han registrado más de 2500 lenguajes de programación diferentes. Diversos autores han realizado una serie de árboles genealógicos a fin de situar los diferentes lenguajes de programación a lo largo de nuestra historia. Además, para desarrollar nuevos lenguajes de programación, los investigadores se han basado en la filosofía de lenguajes ya existentes; de este modo se puede observar cómo algunos lenguajes de programación han heredado ciertas características de sus antecesores. Puesto que es un número muy elevado, y analizar cada uno de los lenguajes de programación de la historia se aleja del objetivo de este curso, se anima al lector a consultar la referencia:

https://web.archive.org/web/20160506170543/http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm

En lugar de centrarnos en los diferentes lenguajes de programación que se han desarrollado a lo largo de la historia, resulta más interesante realizar un estudio sobre cómo las características y funcionalidades de estos lenguajes de programación han evolucionado, tanto para satisfacer las necesidades de los usuarios, como para adaptarse a los nuevos avances tecnológicos. En la siguiente sección estudiaremos los diferentes tipos de lenguajes de programación, así como sus características.

Tipos de lenguajes de programación

Lenguajes procedurales

Los lenguajes de programación procedurales se utilizan para ejecutar una secuencia de declaraciones que conducen a un resultado. Normalmente, este tipo de lenguaje de programación utiliza variables, bucles y otros elementos que los diferencian de los lenguajes de programación funcionales. Algunos lenguajes de programación procedurales más destacados son: FORTRAN, PASCAL, C, ADA, ALGOL.

Las principales ventajas de estos lenguajes de programación son:

- La organización de un programa en procedimientos proporciona una mayor comprensión del código al programador, facilitando las tareas del diseño, depuración y mantenimiento.
- El hecho de que un programa pueda ser desarrollado en módulos independientes permite que unos equipos de personas puedan trabajar en cada módulo por separado.
- Los códigos implementados bajo esta modalidad pueden ser compilados y almacenados en una librería. Las librerías pueden ser importadas por otros programas y beneficiarse de código ya implementado.

Lenguajes funcionales

La programación funcional es un paradigma de le programación declarativa basado en el uso de funciones matemáticas. Estos lenguajes de programación se componen únicamente por definiciones de funciones, entendiéndolas como funciones matemáticas y no como subprogramas encapsulados en una función. Algunos ejemplos de lenguajes de programación funcionales son:

- **Lisp**
- **Haskell**
- **F#**
- **Scala**

Importante

La programación funcional permite un alto grado de abstracción, ya que se basa en el principio de función. Utilizados de forma correcta, este tipo de programación crea códigos muy precisos.

En los problemas donde se abordan tareas matemáticas y algoritmos complejos los lenguajes de programación funcional ocupan una posición estratégica en la informática.

Lenguaje de programación orientados a objetos

Es un paradigma de la programación basado en el que introduce el concepto de objetos. Estos objetos contienen datos en forma de campos (también conocido como atributos y propiedades) y código, también conocido como métodos.

En la programación orientada a objetos, los programas son diseñados de modo que los objetos puedan interactuar unos con otros, pudiendo acceder y modificar (a veces) los valores de sus propios campos o los de otros objetos.

Algunos ejemplos de los lenguajes de programación orientados a objetos más utilizados son:

- **Java**
- **C++**
- **Python**
- **C#**
- **Scala**

Lenguaje de scripting

Son una secuencia de comandos que suelen utilizarse para automatizar tareas repetitivas, hacer procesamiento de lotes e interactuar con el sistema operativo. El lenguaje de programación scripting o scripts no se compilan (para traducir el código a lenguaje máquina), sino que hace uso de un intérprete que se encarga de leer

y traducir cada línea del lenguaje de programación a medida que va siendo ejecutado.

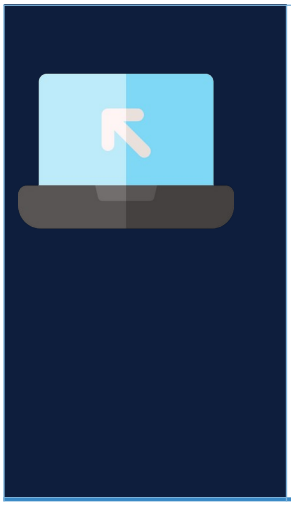
Algunos de los lenguajes de scripting más destacados son:

- **Python**
- **PHP**
- **Ruby**
- **Unix Shell scripts**

Lenguajes de programación compilados e interpretados

Tras conocer la tipología de los lenguajes de programación, es necesario definir qué son los lenguajes de programación compilados e interpretados, así como sus principales ventajas e inconvenientes.

Cuando hablamos de un lenguaje de programación compilado hacemos referencia a aquellos lenguajes que utilizan un compilador para traducir el código escrito por el usuario a código máquina. Este compilador no es más que un programa que se encarga de traducir todo el código a código máquina, una vez hecho esto, el programa puede ser ejecutado. Por otro lado, en los lenguajes de programación interpretados no se hace uso de un compilador, si no que se utiliza un **intérprete**. La principal diferencia radica en que un lenguaje interpretado es convertido a código máquina a medida que éste es ejecutado.



Para más información

Puede consultar el siguiente enlace para conocer más detalles de los compiladores e intérpretes:
https://medium.com/@rahul77349/difference-between-compiler-and-interpreter-with-respect-to-jvm-java-virtual-machine-and-pvm-22fc77ae0eb7

Por mostrar un ejemplo, un lenguaje de programación compilado sería C++ y un fragmento de código asociado a este lenguaje de programación es el que se muestra en la siguiente imagen.

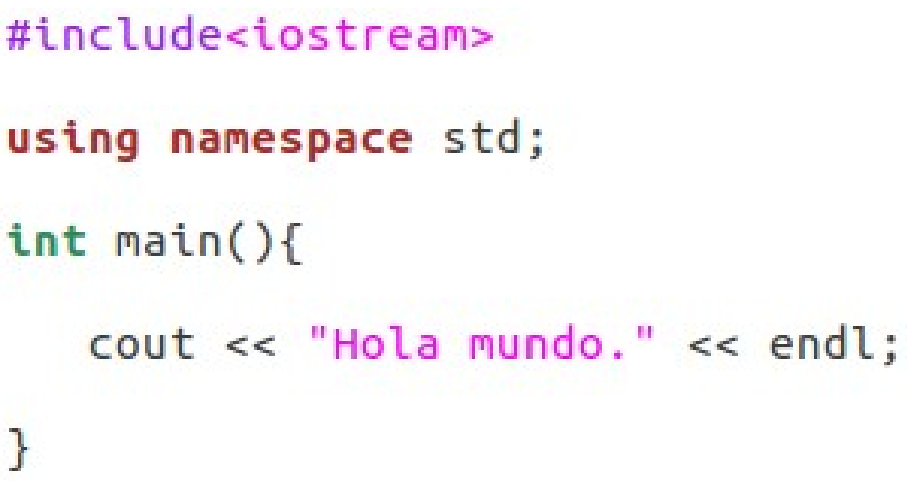


Imagen 4: Ejemplo lenguaje de programación compilado (C++)

Este es un programa sencillo que muestra por pantalla el famoso mensaje "Hola mundo". Para poder ejecutar este programa es necesario utilizar un compilador de C++, como g++. Como hemos comentado anteriormente, este compilador se encarga de convertir el código que hemos implementado para mostrar un mensaje por pantalla a código máquina, entendible por la CPU. Para compilarlo, hemos ejecutado la siguiente instrucción en una terminal:

g++ -o ejecutable nombrePrograma.cpp

Tras compilar este programa, se nos han creado dos nuevos ficheros. Un fichero con la extensión ".o", que se refiere al código máquina que ha generado el compilador y otro fichero ejecutable que será el que utilicemos para ejecutar nuestro programa:

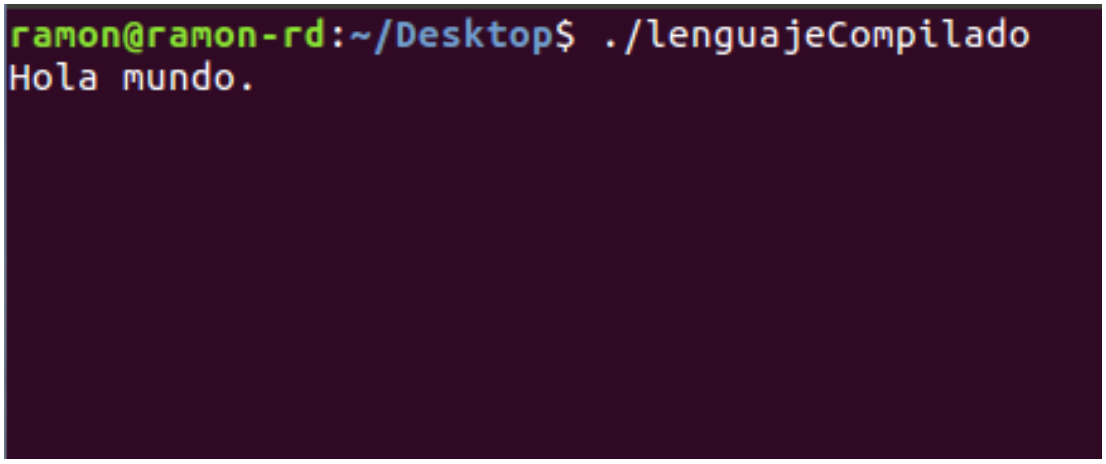


Imagen 5: Salida tras ejecutar el programa implementado en C++.

Cabe destacar que este ejemplo ha sido implementado, compilado y ejecutado en un sistema Linux (Ubuntu 16.04). Esto quiere decir que el programa ejecutable que nos ha generado el compilador solo podrá ser utilizado en sistemas Linux, mientras que, si tratamos de utilizar este programa en un sistema Windows, no será posible (y viceversa, si el programa hubiese sido compilado en un sistema Windows, el programa no podría utilizarse en un sistema Linux). Esta es una de las principales limitaciones de los lenguajes de programación compilados.

Por otro lado, un ejemplo de lenguaje interpretado sería el lenguaje de programación **Python**. Como hemos estudiado, al ser un lenguaje de programación interpretado, no es necesario utilizar un compilador para traducir el código a código máquina. El propio intérprete de Python se encarga de transformar el programa a medida que va ejecutándolo. Un ejemplo del mismo programa implementado en C++, pero en el lenguaje Python sería el siguiente:

```
print("Hola mundo")
```

Imagen 6: Ejemplo lenguaje de programación interpretado (Python)

Tras implementar este pequeño programa de prueba, bastaría con ejecutarlo como sigue:

```
ramon@ramon-rd:~/Desktop$ python lenguajeInterpretado.py
Hola mundo
ramon@ramon-rd:~/Desktop$
```

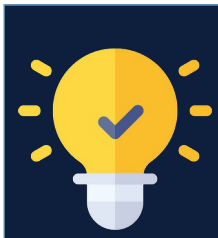
Imagen 7: Salida tras ejecutar el programa implementado en Python.

En este caso, a diferencia del lenguaje de programación compilado, podemos portar la aplicación a cualquier sistema operativo, y podría ser ejecutada (siempre y cuando el sistema operativo donde vayamos a ejecutar el código tenga instalado una versión compatible de Python).

Presta atención

En términos de diseño, implementación y prueba de un algoritmo, un lenguaje de programación interpretado es más rápido, en comparación con un lenguaje de programación compilado. Este se debe a que cada vez que implementamos un fragmento de código y queremos validarlo, usando un lenguaje de programación compilado, debemos compilar el programa tantas veces como pruebas queramos realizar. Sin embargo, utilizando un lenguaje de programación interpretado, bastaría con ejecutar el código en una terminal, sin necesidad de hacer uso de un compilador.

No obstante, un lenguaje compilado es mucho más rápido que uno interpretado. Esto se debe a que el programa que se ejecuta como resultado de compilar un código, ya se encuentra traducido a código máquina, lo que también posibilita realizar algunas optimizaciones que no son posibles con un lenguaje interpretado. Además de dicha velocidad, un programa compilado puede ser ejecutado en cualquier sistema operativo compatible, sin embargo, en un lenguaje interpretado es necesario instalar un intérprete en el sistema operativo donde será ejecutado el programa, para que pueda ser utilizado.

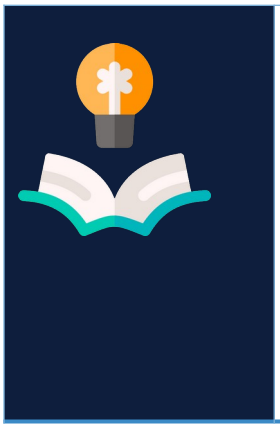


Importante

Un lenguaje compilado está optimizado para el momento de la ejecución, a costa de una carga adicional para el programador.
Un lenguaje interpretado está optimado para facilitar la tarea al programador, a costa de una carga adicional de la máquina.

Historia y filosofía de Python

El lenguaje de programación Python fue creado por Guido Van Rossum en el centro de las Matemáticas y la Informática a finales de 1980. Python surge como sucesor del lenguaje ABC, capaz de manejar excepciones e interactuar con el sistema operativo Amoeba. El nombre asociado a este lenguaje de programación proviene por la afición de Guido por los humoristas británicos Monty Python.



Presta atención

Existen tres versiones principales de Python:

Python 1.0 que fue liberado al público en 1994, Python 2.0 en 2000 y Python 3.0 en 2008. Cabe destacar que la versión 3 de Python no es compatible con versiones anteriores.

Python se ha establecido como uno de los lenguajes de programación optimizamos y bien establecidos como C, C++ y Java principalmente por su filosofía. Python fue diseñado con la filosofía de que las tareas complejas puedan ser realizadas de forma sencilla. Python fue creado para tener una curva de aprendizaje y un proceso de desarrollo muy rápido y simple para los ingenieros de software. Como resultado, se considera el lenguaje de programación de propósito más general, ya que los usuarios pueden trabajar en casi cualquier dominio de estudio y aun así encontrar código útil para ellos mismos. Python aprovechó el poder del movimiento de código abierto, lo que le ayudó a atraer una gran cantidad de usuarios de diversas especialidades. Al ser de código abierto, Python permitió a los desarrolladores crear pequeños programas y compartirlos entre sí con facilidad. La gran cantidad de módulos y desarrolladores ha permitido que el uso de Python se haya convertido en una herramienta principal en el área de ciencias de la

computación y que finalmente, sea el lenguaje de programación preferido por los desarrolladores.

Importante

Python es un lenguaje de programación multipropósito, portable, orientado a objetos y de alto nivel que permite desarrollar código de una manera sencilla, elegante y legible.

Las principales características de Python son las siguientes:

- Es un lenguaje de programación interpretado y de alto nivel lo que facilita el uso de funciones para desarrollar tareas de bajo nivel y además proporciona mecanismos para implementar código de una forma legible para los humanos, que luego se traduce a lenguaje máquina por medio de un intérprete.
- Es un lenguaje de programación orientado a objetos. La naturaleza abstracta de los objetos permite a los usuarios diseñar y desarrollar objetos de acuerdo con sus necesidades, y aplicar conceptos de programación para una gran variedad de aplicaciones.
- Debido a su naturaleza orientada a objetos, Python se ha convertido en un lenguaje de programación muy versátil, ya que los programadores han desarrollado módulos con el objetivo de facilitar la resolución de trabajos en su área de especialización. Esto implica que una gran cantidad de áreas han utilizado Python, y ahora la comunidad de desarrolladores cuentan con una gran librería útil para resolver problemas de distinta índole, como:

    - Cálculo científico
    - Estadística
    - Criptografía
    - Desarrollo de videojuegos
    - Bases de datos
    - Procesamiento de imágenes

- Procesamiento de lenguaje natural
- Y un largo etcétera.

- Una de sus características más diferenciadores es que está definido para tener un diseño minimalista y que su código sea altamente legible. Además, proporciona una sintaxis que permite desarrollar programas utilizando menos líneas de código que en lenguajes como Java y C++.
- Puesto que Python pertenece a la comunidad de código abierto, este ha sido portado a una gran variedad de plataformas, entre las que destacan Linux, Windows, Macintosh o Solaris. La principal ventaja es que un programador que ha desarrollado un código en Linux puede ser utilizado, modificado y actualizado por otro programador que utiliza Windows.
- Python fue desarrollado para ser extensible lo que implica que sus usuarios pueden optar por utilizar una determinada funcionalidad atendiendo a sus necesidades. Por ejemplo, si un usuario necesita trabajar con bases de datos, entonces puede cargar y utilizar el módulo especializado en bases de datos, en lugar de que todos los usuarios dispongan de esta funcionalidad y que nunca sea utilizada.

Presta atención

Python permite integrar otros códigos de diferentes lenguajes de programación como R o C++, en su propio código. Esta función ha permitido el uso de una gran cantidad de código heredado escrito en oro lenguajes, evitando la duplicación de código y aumentando la productividad de desarrollo.

Para más información

Puede consultar más detalles sobre la filosofía de Python en el documento “The Zen of Python” desarrollado por Tim Peters.

¡Enhorabuena!

Has finalizado el estudio de esta unidad